Hardware Performance Monitoring and Dynamic Instrumentation

Shirley Moore
shirley@cs.utk.edu

Alliance Performance Team Workshop

Urbana, IL March 14, 2002

PAPI Development Team

- Philip Mucci, Technical Lead
- Jack Dongarra
- Kevin London
- Daniel Terpstra
- Haihang You

Hardware Counters

- Small set of registers that count *events*, which are occurrences of specific signals related to the processor’s function
- Monitoring these events facilitates correlation between the structure of the source/object code and the efficiency of the mapping of that code to the underlying architecture.

Overview of PAPI

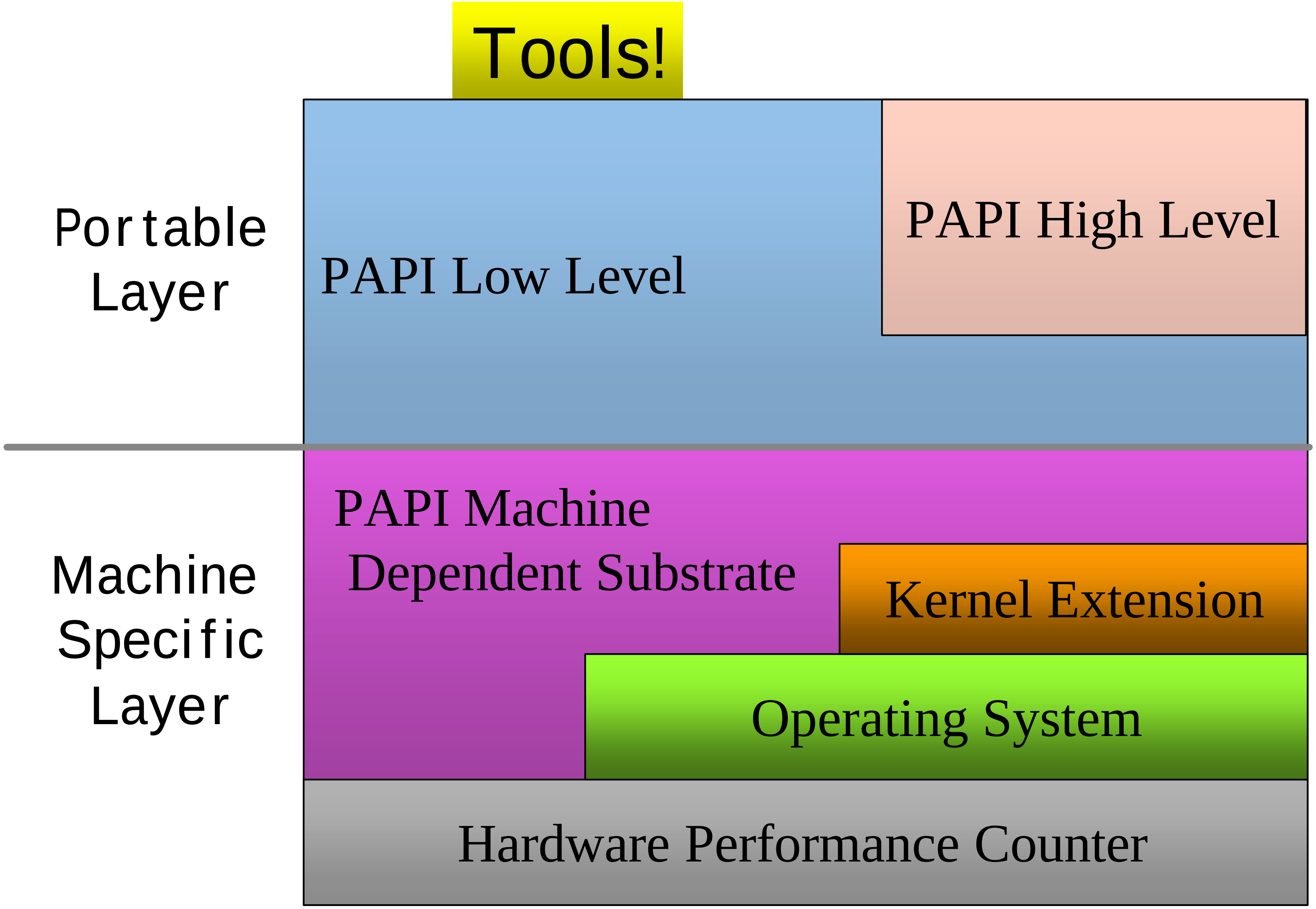
- **P**erformance **A**pplication **P**rogramming **I**nterface
- The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- Parallel Tools Consortium project

  http://www.ptools.org/

PAPI Counter Interfaces

- PAPI provides three interfaces to the underlying counter hardware:
  1. The low level interface manages hardware events in user defined groups called *EventSets*.
  2. The high level interface simply provides the ability to start, stop and read the counters for a specified list of events.
  3. Graphical tools to visualize information.

PAPI **Implementation**



PAPI Preset Events

- Proposed standard set of events deemed most relevant for application performance tuning
- Defined in papiStdEventDefs.h
- Mapped to native events on a given platform
  - Run tests/avail to see list of PAPI preset events available on a platform

PAPI 2.1 Release

- Platforms
  - Linux/x86, Windows 2000
    - Requires patch to Linux kernel, driver for Windows
  - Linux/IA-64
  - Sun Solaris/Ultra 2.8
  - IBM AIX/Power
    - Requires pmtoolkit (available from http://alphaworks.ibm.com/)b
  - SGI IRIX/MIPS
  - Cray T3E/Unicos
- Fortran and C binding and MATLAB wrappers

High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Not thread safe
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (additional code) than low-level

High-level API

- C interface
  PAPI_start_counters
  PAPI_read_counters
  PAPI_stop_counters
  PAPI_accum_counters
  PAPI_num_counters
  PAPI_flops

- Fortran interface
  PAPIF_start_counters
  PAPIF_read_counters
  PAPIF_stop_counters
  PAPIF_accum_counters
  PAPIF_num_counters
  PAPIF_flops

PAPI_flops

- int PAPI_flops(float *real_time, float *proc_time, long_long *flpins, float *mflops)
  - Only two calls needed, PAPI_flops before and after the code you want to monitor
  - real_time is the wall-clocktime between the two calls
  - proc_time is the “virtual” time or time the process was actually executing between the two calls (not as fine grained as real_time but better for longer measurements)
  - flpins is the total floating point instructions executed between the two calls
  - mflops is the Mflop/s rating between the two calls

PAPI High-level Example

```
long long values[NUM_EVENTS];
unsigned int
  Events[NUM_EVENTS]={PAPI_TOT_INS,PAPI_TOT_CYC};
 /* Start the counters */
 PAPI_start_counters((int*)Events,NUM_EVENTS);
 /* What we are monitoring? */
 do_work();
 /* Stop the counters and store the results in values */
 retval = PAPI_stop_counters(values,NUM_EVENTS);
```

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- About 40 functions
- Obtain information about the executable and the hardware
- Thread-safe
- Fully programmable
- Callbacks on counter overflow

Low-level Functionality

- Library initialization

  PAPI_library_init, PAPI_thread_init, PAPI_shutdown

- Timing functions

  PAPI_get_real_usec, PAPI_get_virt_usec PAPI_get_real_cyc, PAPI_get_virt_cyc

- Inquiry functions
- Management functions
- Simple lock

  PAPI_lock/PAPI_unlock

Event sets

- The event set contains key information
  - What low-level hardware counters to use
  - Most recently read counter values
  - The state of the event set (running/not running)
  - Option settings (e.g., domain, granularity, overflow, profiling)
- Event sets can overlap if they map to the same hardware counter set-up.
  - Allows inclusive/exclusive measurements

Event set Operations

- Event set management PAPI_create_eventset, PAPI_add_event[s], PAPI_rem_event[s], PAPI_destroy_eventset
- Event set control PAPI_start, PAPI_stop, PAPI_read, PAPI_accum
- Event set inquiry PAPI_query_event, PAPI_list_events,...

Simple Example

```
#include "papi.h“
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC}, EventSet;
   long_long values[NUM_EVENTS];
/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* Add Flops and total cycles to the eventset */
retval = PAPI_add_events(&EventSet,Events,NUM_EVENTS);
/* Start the counters */
retval = PAPI_start(EventSet);

do_work();  /* What we want to monitor*/

/*Stop counters and store results in values */
retval = PAPI_stop(EventSet,values);
```

Using PAPI with Threads

- After PAPI_library_init need to register unique thread identifier function
- For Pthreads

```
retval=PAPI_thread_init(pthread_self, 0);
```

- OpenMP

```
retval=PAPI_thread_init(omp_get_thread_num, 0);
```

- Each thread responsible for creation, start, stop and read of its own counters

Using PAPI with Multiplexing

- Multiplexing allows simultaneous use of more counters than are supported by the hardware.
- PAPI_multiplex_init()
  - should be called after PAPI_library_init() to initialize multiplexing
- PAPI_set_multiplex(int *EventSet);
  - Used after the eventset is created to turn on multiplexing for that eventset
- Then use PAPI like normal

Issues with Multiplexing

- Some platforms support hardware multiplexing, on those that don’t PAPI implements multiplexing in software.
- The more events you multiplex, the more likely the representation is not correct.

Native Events

- An event countable by the CPU can be counted even if there is no matching preset PAPI event
- Same interface as when setting up a preset event, but a CPU-specific bit pattern is used instead of the PAPI event definition

Callbacks on Counter Overflow

- PAPI provides the ability to call user-defined handlers when a specified event exceeds a specified threshold.
- For systems that do not support counter overflow at the OS level, PAPI sets up a high resolution interval timer and installs a timer interrupt handler.

PAPI_overflow

- int PAPI_overflow(int EventSet, int EventCode, int threshold, int flags, PAPI_overflow_handler_t handler)
- Sets up an EventSet such that when it is PAPI_start()'d, it begins to register overflows
- The EventSet may contain multiple events, but only one may be an overflow trigger.

Statistical Profiling

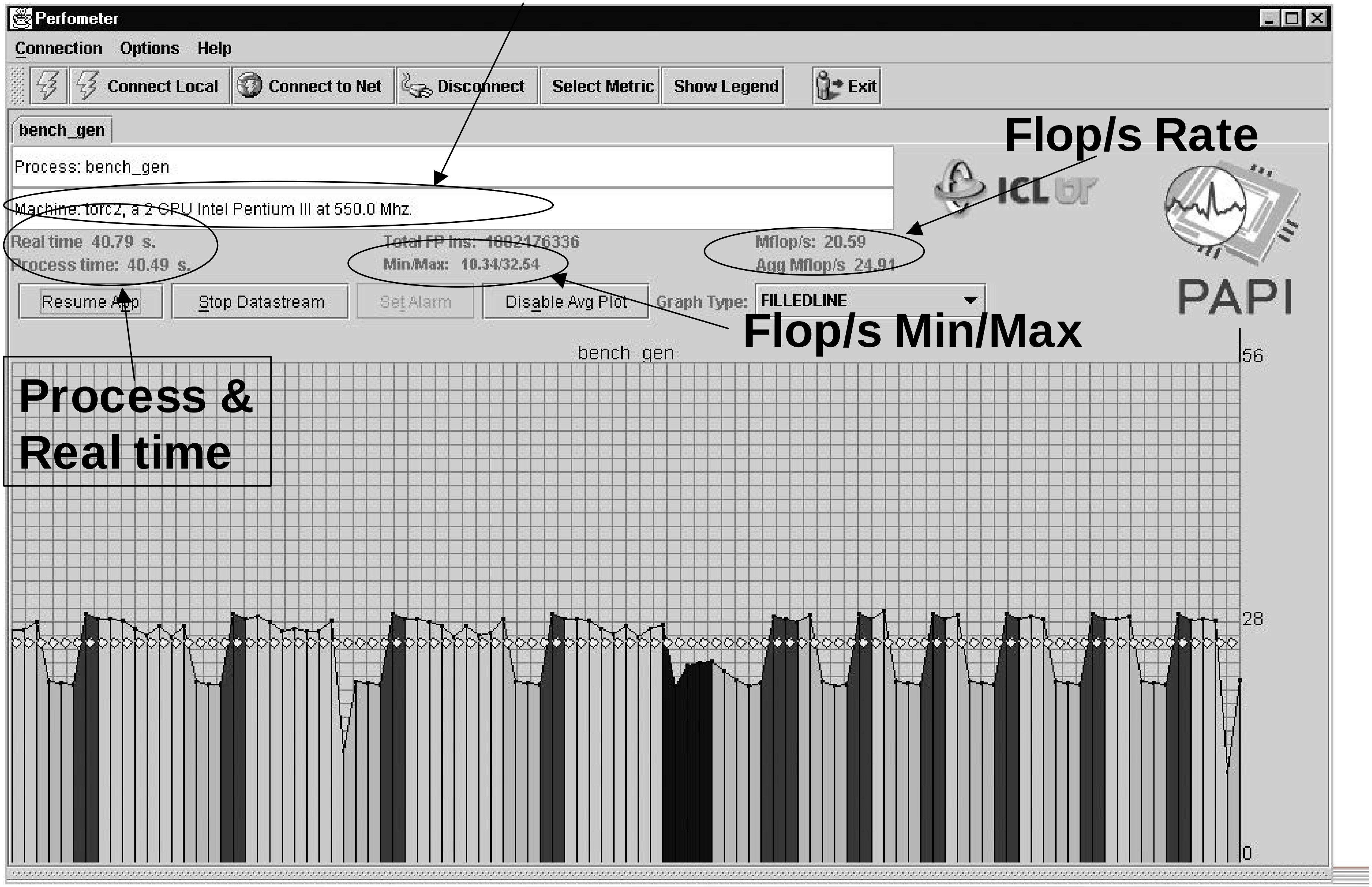
- PAPI provides support for execution profiling based on any counter event.
- PAPI_profil() creates a histogram of overflow counts for a specified region of the application code.

Perfometer

- Application is instrumented with PAPI
  - call perfometer()
  - Call mark_perfometer(Color)
- Application is started. At the call to **perfometer**, signal handler and timer are set to collect and send the information to a Java applet containing the graphical view.
- Sections of code that are of interest can be designated with specific colors
  - Using a call to mark_perfometer('color')
- Real-time display or trace file

Perfometer Display

Machine info



Process: bench_gen

Machine: torc2, a 2 CPU Intel Pentium III at 550.0 Mhz.

Real time 40.79 s.

Process time: 40.49 s.

Total FP Ins: 1002176336

Min/Max: 10.34/32.54

Mflop/s: 20.59

Agg Mflop/s 24.91

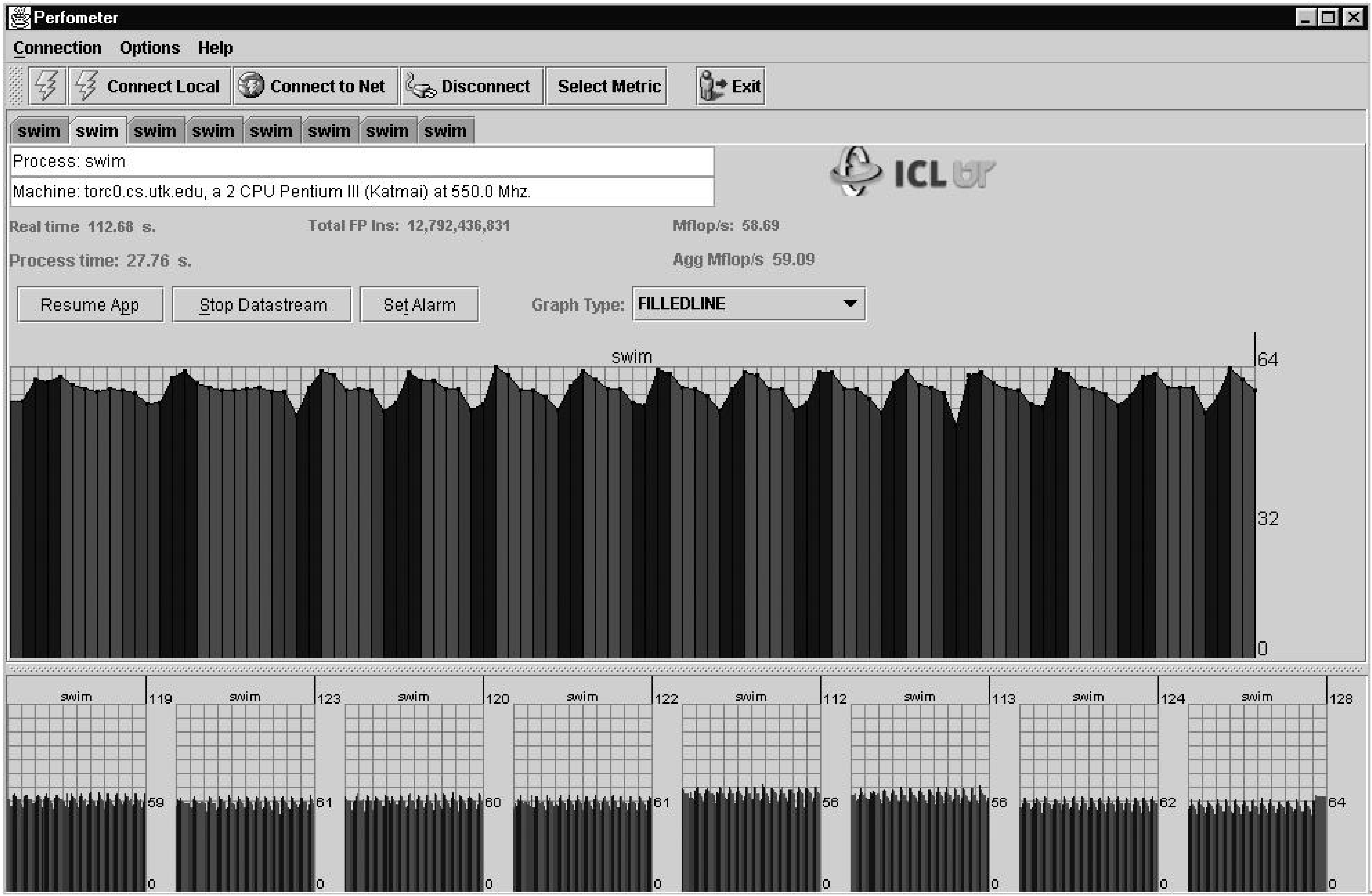
Graph Type: FILLEDLINE

Flop/s Rate

Flop/s Min/Max

Process & Real time

Perfometer Parallel Interface



Process: swim

Machine: torc0.cs.utk.edu, a 2 CPU Pentium III (Katmai) at 550.0 Mhz.

Real time 112.68 s.

Total FP Ins: 12,792,436,831

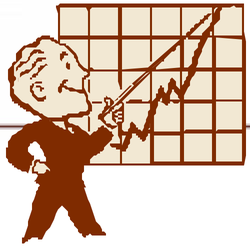
Mflop/s: 58.69

Process time: 27.76 s.

Agg Mflop/s 59.09

Graph Type: FILLEDLINE

Third-party Tools that use PAPI



- DEEP/PAPI (Pacific Sierra) http://www.psrv.com/deep_papi_top.html
- TAU (Allen Mallony, U of Oregon) http://www.cs.uoregon.edu/research/paracomp/tau/
- SvPablo (Dan Reed, U of Illinois) http://vibes.cs.uiuc.edu/Software/SvPablo/svPablo.htm
- Scalea (Thomas Fahringer, U. Vienna)

  http://www.par.univie.ac.at/project/scalea/
- Vprof (Curtis Janssen, Sandia Livermore Lab) http://aros.ca.sandia.gov/~cljanss/perf/vprof/
- Cluster Tools (Al Geist, ORNL)
- DynaProf (Phil Mucci, UTK) http://www.cs.utk.edu/~mucci/dynaprof/

DynaProf

An Object Code Instrumentation System for Dynamic Profiling

Philip J. Mucci mucci@cs.utk.edu November, 2001

What is DynaProf?

- A portable tool to instrument a running executable with *Probes* that monitor application performance.
- Simple command line interface.
- Open Source Software
- A work in progress...

No source code required

DynaProf Methodology

- Make collection of run-time performance data easy by:
  - Avoiding instrumentation and recompilation
  - Using the same tool with different probes
  - Providing useful and meaningful probe data
  - Providing different kinds of probes
  - Allowing custom probes

No source code required!

Why the "Dyna"?

- Instrumentation is selectively inserted directly into the program's address space.
- Why is this a better way?
  - No perturbation of compiler optimizations
  - Complete language independence
  - Multiple Insert/Remove instrumentation cycles

DynaProf Design

- GUI, command line & script driven user interface
- Uses GNU readline for command line editing and command completion.
- Instrumentation is done using:
  - Dyninst on Linux, Solaris and IRIX
  - DPCL on AIX

DynaProf Commands

load <executable>

list [module pattern]

use <probe> [probe args]

instr module <module> [probe args]

instr function <module> <function> [probe args]

stop

continue

run [args]

Info

unload

Dynaprof Probes

- papiprobe
- wallclockprobe
- perfometerprobe

DynaProf Probe Design

- Can be written in any compiled language
- Probes export 3 functions with a standardized interface.
- Easy to roll your own (<1day)
- Supports separate probes for MPI/OpenMP/Pthreads

Future development

- GUI development
- Additional probes
  - Perfex probe
  - Vprof probe
- Better support for parallel applications

For More Information

- http://icl.cs.utk.edu/papi/
  - Software and documentation
  - Reference materials
  - Papers and presentations
  - Third-party tools
  - Mailing lists

- http://www.ncsa.uiuc.edu/UserInfo/Resources/Software/Tools/PAPI/

Current and Future Work

- Ports – P4, Power4, McKinley, Compaq Alpha
- Accuracy and efficiency issues
- Infrastructure for dynamic instrumentation of parallel applications (DPCL?)
- Experimentation with IA-64 performance monitoring features (e.g., event qualification, EARs)